```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                                       *
*                        BASIC/S II COMPILER                            *
*                                                                       *
*          (C) 1982 by Bill Stockwell and Breeze/QSD                    *
*                -Version 1.5 for Mod I and III-                        *
*                     -All Rights Reserved-                             *
*                                                                       *
*     Published by: Breeze/QSD, Inc., Dallas, Texas                     *
*                                                                       *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

Getting Started Using BASIC/S II:

IMPORTANT!! There is a variable in BASIC/S which tells BASIC/S what disk operating system you are using. Currently, this is used so that LOF calculations will be done properly - ie when you compile a program that does an LOF calculation, it is important for the compiler to know what DOS is being used so that this calculation will be done properly. (The assumption is that you will run your /CMD files under the same system that they were compiled on. If this is not true, you can change the value of this variable as explained below and recompile under the other DOS). This is discussed under "Options", when you execute BASICSII.

On the disk you receive, there will be just one copy of BASICSII/CMD, one of COMPILE/DAT, and some supplementary demo and utility files. Copy these to a disk of your own. One of these files is REMPER/BAS, a utility useful for those who have programs written for the original BASIC/S (the original BASIC/S requires percent signs after integer variable names, whereas BASICSII regards A-Z as integer variables - no percent signs allowed!) REMPER will remove all percent signs from an ascii BASIC file, so that it will now be compilable by BASICSII so long as no real variables are used in it.

Another file is QUICK/BAS, which generates an integer array and does a quicksort on it, and prints out the results.

Also there is BINHEX/BAS, originally by Tim Mann and rewritten by Bill Stockwell into BASIC/S compilable form. This program is for converting HEX files to/from /CMD file format. HEX files are the typical way in which binary files are stored on bulletin board systems for transfer via modem.

Finally on the disk is a utility for allowing BASICSII to handle floating point values (in a limited way) (see the files FLOAT/TXT, FLOAT/BAS, and SQR/BAS for more info on this).

The version of BASIC which is supported is a subset of Disk Basic. Only simple expressions and variable names are allowed, but most of the features and built-in functions of Level II are implemented, along with the essential elements of sequential and random disk I/O. Floating point variables are not supported (BASICSII/CMD is 40K as it is!), but integers, strings, and arrays of

type integer or string are allowed. Note: Unlike regular BASIC, programs compiled by BASIC/S do NOT have any initialization of variables done. Thus numeric variables do not start out as zero, or strings as null. (See the CLEAR statement, however). One advantage of this approach is that one compiled program can invoke another (using the RUN statement) and all variables will be preserved.

Use of constants in BASIC/S is somewhat restricted; many statements allow (integer) constants ; most statements do NOT allow string constansts. See the section below on the individual statements for more details.

You may have multiple statements per line; the only restriction here is that IF, GOTO, and GOSUB statements must begin the line they are on (as must ON GOTO).

Look over some of the sample programs on the disk to see how statements are to be coded. The syntax must be followed....

-------> E X A C T L Y !! <-------

...however, the spacing is up to you. Thus, you could say FOR I=1 TO N or you could say FORI=1TON. The compiler allows the following variable names (all single letters): integers A thru Z and strings A$ thru Z$. Also, you may dimension arrays (of either type), and your array names can be any length, with every character significant. See the DIM statement for more on this.

REQUIREMENTS :

A TRS-80(c) Mod I or III with at least one drive and 48K.
Repeat... 48K! Two drives are certainly preferable.

RUNNING THE PROGRAM :

Just type BASICSII from Dos Ready. Be sure that you do not have too much in high memory -- if HIGH$ is less than FE00H, the compiler may run out of string space (or other strange errors may occur). If you are running Mod I LDOS, you can run Lower Case and PDUBL, but not much else in high memory. For best results, run with HIGH$ (HIMEM) = FFFF, if possible.
(Note on LDOS 5.1 for Mod I - you may have PDUBL and KI/DVR in high memory - nothing else - to use BASIC/S II. When you SET KI/DVR, do NOT use type ahead).

After BASICSII begins, you may remove the disk containing it, and insert the one with COMPILE/DAT, if necessary, (if you have only one drive, this must be a SYSTEM disk). Remember -- COMPILE/DAT must be

(c)1982 by Breeze/QSD, Inc.

on line AT ALL TIMES while you compile, as well as the program you wish to compile (saved in ASCII!).

REPEAT.... your program to compile MUST be saved in ASCII!

e.g. SAVE"BASPROGM/BAS",A <enter>

(the /BAS extension is NOT required)

Now BASIC/S will ask:

```
Source :
Object :
Options :
```

(one after another). Typically, you will answer the first two questions, and hit <enter> on the last one. "Source" is the name of the file to be compiled, and "Object" is the name of the /CMD file you want to create from "Source". Thus if you had an ASCII BASIC program called TEST/BAS that you wanted to compile, you might answer the above with:

```
Source : TEST/BAS
Object : TEST/CMD:2
Options : <enter>
```

The options you might take are as follows :

You can specify Start Address, whether or not to list the source file to the video during the compile, whether to disable the <break> key (for the compiled code), and what DOS is being used. Any or all may be specified, but those that are used should be in the correct order. The start address tells BASICSII where in memory your object file should load to -- the default being 5200H (20992 decimal). The address MUST be a decimal integer, but can be positive or negative; ie D6D8H is represented by either 55000 or -10536; BASIC/S knows what you mean.

Use the letter N to indicate No list - BASIC/S normally lists your source file to the video as it compiles, but if you don't want this, just answer Options : with 'N' (after the address, if there is one). You might want to do this if you were getting a lot of errors which were scrolling off the screen too fast.

You can specify the DOS you are using via:
S=x (x being an integer value).

Use:
4 for DOSPLUS 3.3
5 for LDOS
1 for Newdos/80 and DOS+ 3.4
0 for any other DOS
The default is 5 (LDOS).

(c)1982 by Breeze/QSD, Inc.

TRSDOS (Mod III) is _N_O_T__S_U_P_P_O_R_T_E_D.

To start the /CMD file at 56000, with no listing, and using Mod I TRSDOS, you would answer 'Options' with

Options : 56000,N,S=0

Note that the S= option is important ONLY if your source file computes LOF's. Also note that BASIC/S accepts lower case responses.

A new option, added in version 1.1 of BASIC/S II, allows you to disable the break key while the BASIC/S /CMD file that is created executes. This is done by typing the letter "B" among your options - like the other options, this must come AFTER any address. Doing this will cause the following to occur: during input, while a BASIC/S /CMD file executes with the "B" option, if the <break> key is typed, then a line feed is done and the input starts over at the beginning.

THE BASIC/S SUBSET (Statements supported under BASIC/S) :

PRINT

followed by a SINGLE variable name, or an expression in quotes. Thus :

```
PRINT A or
PRINT"Message" or
PRINT R$
```

Also, you may use a semi-colon after anything being printed in order to suppress the carriage return.

PRINT@ is also supported -- just set any integer variable (or constant) to the value of the location to be printed at, and you may then use any of the above forms with it. Thus :

```
PRINT@N,"TRS-80";
```

It is important to note that you may NOT print a list of items when using BASIC/S; only one item (of any type) may be PRINTed at a time. The same applies to INPUT.

LPRINT

Syntax for LPRINT is in every way the same as for PRINT, except of course that LPRINT@ has no meaning.

(c)1982 by Breeze/QSD, Inc.

INPUT

You may input a single variable, of any type. You may not input a list of variables, but INPUT"PROMPT";A is supported (or A$).
When executing a Basic/s compiled program, if input is requested, hitting the <break> key will cause an exit
If in answer to an input prompt, you hit <Enter> only, then the variable being inputted remains unchanged and the program continues (just like regular BASIC -- and this holds regardless of variable type.

LINE INPUT

LINE INPUT from the keyboard is supported. Syntax is exactly as it is in BASIC. You may LINE INPUT an integer variable if you wish, although this would not work in BASIC.

e.g. LINE INPUT A$ or
LINE INPUT "Prompt";A$ (just like in BASIC)

RUN A$

This statement allows you to set a string (A$ in this case) to any DOS command, or the name of a command file you wish to invoke, and to exit the current program and have that command executed. Do NOT say RUN"PGM"; this will be not be correctly compiled! Also, RUN by itself is incorrect.

DEFINT

For compatibility with the BASIC interpreter, you may use this statement (as in DEFINT A-Z). No other DEF statements are accepted, and this one only reaffirms what BASIC/S II does anyway - regards all variables as integers unless they are suffixed with '$'.

CLEAR

This statement, with or without an argument, will cause BASIC/S variables to be zeroed out. It depends on where your /CMD file starts; if your /CMD file is in low memory, then all memory from 41216 (decimal) up to HIGH$ will be zeroed out, while otherwise 5200H up to D6D8H is zeroed out. This makes sure that your /CMD file itself will never be affected, but that your variables will be zeroed. This works equally well on

(c)1982 by Breeze/QSD, Inc.

the Mod I or the Mod III - BASIC/S knows which machine you are running it on, and will use the correct HIGH$ for your machine. DATA will also be cleared, and an automatic RESTORE done so that the DATA pointer will be correct.

GOTO ln

The GOTO statement -- works just as in BASIC, but it MUST BEGIN the line it is on. Thus CLS:GOTO 20 will not work, although no message would be given.

ON GOTO

As in BASIC, except that the index must be a simple variable (not an expression). Thus

```
ON X GOTO 20,30,1000
```

No limit on the number of different lines you can branch to, other than the limitation of 255 chars per line. ON GOSUB is NOT supported. Like IF, GOTO, and GOSUB, ON GOTO statements must begin the line they are on.

GOSUB ln

The standard GOSUB statement, but like GOTO, must begin the line it is on.

READ / DATA / RESTORE

Your program may have DATA statements, containing integer constants only (as in DATA 1,2,3) -- in all of your DATA statements you can have a total of 383 integers (no more). It is important that these DATA statements come before the READ statement(s) that are to access them (physically before, that is) -- the compiler generates code to place the data in memory when the DATA statements are encountered. Syntax for the READ statement is READ N -- you can read only a single integer variable, which would normally be done in a FOR/TO loop. One big use for this is to poke DATA for a USR routine into memory. Before BASIC/S allowed READ/DATA, this process was rather clumsy.

RESTORE
---

works just like in standard BASIC.

(c)1982 by Breeze/QSD, Inc.

IF

A very restricted IF statement -- you may only compare two strings (for equality or in the < direction), or two simple integers (variables or constants). Thus (for strings) :

```
IF A$<B$ THEN 20
or      IF A$=B$ THEN 100
```

The compare must be in the '<' direction only, or with '='. You may check whether a string is null via

```
IF A$="" THEN 200 (for example)
```

but this is the only time you may test a string against a constant.

For integers :

```
IF A=B THEN 100
or      IF A<B THEN 50
```

(and either A or B may be an integer constant, as in IF A<72 THEN 200).

*** Note: GOTO, GOSUB, and IF statements MUST begin the line that they are on. Also, ELSE is now supported:
you may follow any IF statement with ELSE, followed by as many statements as you can fit on one line, so long as they do not need to start the line they are on. Thus IF, GOTO, GOSUB, and ON GOTO statements may not follow an ELSE, but any other statement may do so.

FOR/NEXT

The For/Next loop is implemented for INTEGERS only. You may code

```
FOR A=B TO C
  ...
NEXT A
```

Constants may be used where B and C are indicated, as long as they are integers (positive, negative, or zero).
The variable in the NEXT statement is NOT optional.
There is no STEP clause.
FOR/NEXT loops may be (statically) nested.

USR

A single USR call is allowed. It must be set up by

(c)1982 by Breeze/QSD, Inc.

DEFUSR, and the calling address must be a simple decimal integer constant. Thus :

```
DEFUSR=-1000
```

Note: There is no VARPTR statement. However, the addresses of all simple variables in BASIC/S are always the same and may be calculated as follows :

If A is the ascii code of the variable in question, then the VARPTR is :

```
INTEGERS : -11406 + 2 * (A - 65);
STRINGS  : -23192 + 256 * (A - 65).
```

Strings are stored a little differently than in Level II. Each string is allocated 256 bytes, the first of which contains the length of the string (0 to 255) and the rest of which contain the string itself. The Varptr points to the length byte.

Y=USR(X)

This causes the routine whose address was defined by a previous DEFUSR statement to be called. The current value of X is loaded into the HL register pair before the call is made, and on return, Y is given the value in the HL register pair. Do NOT call the ROM routines at 0A7F and 0A9A for this. Any integer variables may be used, not just X and Y. Also, a (decimal) integer constant may be used as the argument to be passed.

SET, RESET, and POINT

Use integers (either variables (followed by) or constants) as the arguments. As with most BASIC/S functions, they may not be used in more complex expressions. Thus

```
SET(X,20)
A=POINT(B,C)
```

The latter is the only way to access POINT - it cannot be invoked in an IF statement.

PEEK and POKE

Exactly as in Level II, except that the arguments must be integers -- (constants or variables). Thus

```
A=PEEK(M)
```

(c)1982 by Breeze/QSD, Inc.

```
POKE A,B
POKE 15360,191
Z=PEEK(14312)
```

INP and OUT

Syntax here is just like that for PEEK and POKE, i.e. you may use integer variables or constants as the arguments (no expressions).

```
A=INP(P) (input a byte from port P and
            store in A)
OUT P,V  (output value V to port P)
OUT 255,1
S=INP(232)
```

AND/OR

You may use these two functions in order to calculate an AND/OR result (for integer variables or constants) and store the answer in an integer variable. Thus

```
X=Y AND 20
U=A OR B
```

CLS -- Clear the screen

RND

Random integers between 1 and N may be generated by the statement X=RND(N). The left hand side may be any integer variable. The argument is required and may be an integer constant if you like. The statement RANDOM is also supported, to reseed the random number generator.

DIM

You can DIMension up to 20 arrays in a program to be compiled with BASIC/S - they can be either integer or string,as distinguished by the presence of a $.
Array names may be any length (up to 255) with every character significant. ONLY letters A-Z should be used within an array name. Thus

```
DIM ARRAY(20,7),ST$(15)
```

You may have one or two dimensions for each array - no

(c)1982 by Breeze/QSD, Inc.

more. DO NOT use BASIC keywords in your array names. Be careful about your available array space - BASIC/S will tell you if your array space will overlay BASIC/S data areas or will exceed the 64K memory limit. If this happens, try recompiling with a start address of 56000; this will give you about 19.75 K of space for your arrays, as it puts your /CMD file in high memory instead of low. Still, 19.75K is only enough room for a string array of dimension 79 (79 * 256 = 20,224). With integer arrays, you can use much larger dimensions.

Syntax for using array elements :
For the most part, you can use your array variables just like any other variables; and you may always use integer constants (as well as variables) for the subscripts). Thus

```
READ NUM(I)
INPUT ARRAY(7)
PRINT ST$(U);
A$=LEFT$(ST$(5),NUM(I))
```

The exceptions are as follows :
When an array element is on the left hand side of the '=' sign, the right hand side MUST be a simple variable or constant of the same type - no expressions allowed. Thus ST$(1)=LEFT$(A$,2)is not allowed; you would need to set H$=LEFT$(A$,2) and then ST$(1)=H$. However, it is OK to set an array element to a constant, as in ST$(5)="HELLO" or ARRAY(14,6)=12.
Also, any statement that references an array element should contain NO numeric constants of any kind, except for (possibly) subscripts to the array itself.
One exception here is that array elements may be compared via the IF statement, and the line number reference will not be misconstrued. So

```
IF ST$(1)<ST$(I) THEN 75
```

is OK; just be sure to follow the syntax in all other respects. But something like

```
LINE INPUT#1,ST$(I)
```

or

```
PUT 1,L(I)
```

won't work as the '1' will be misunderstood, and translated to a temporary integer variable, which won't work.
Thus in general, the statements in which you may not reference array elements are most of the DISK I/O statements (OPEN, FIELD, GET, PUT, LINE INPUT#, PRINT#), and PRINT@.

SCAN

This statement allows the user to "scan" a file or device for a single byte (similar to INKEY$ for the keyboard). First you OPEN the

(c)1982 by Breeze/QSD, Inc.

file or device in question for input; then

```
SCAN b,A$
```

will read a byte from the file or device with DCB# b (must be a constant, 0-9) into A$.

SET EOF

(For use with LDOS only). This statement allows you to truncate a random access file at a specified record. If you have a random access file (DCB 1 or 2 only, in BASIC/S) open, then to cause it to have 50 records (instead of say 100), just GET 1,R (where R=50) and then SET EOF1 (exactly as in LBASIC). Of course you need to close the file to make sure the directory entry is updated. This could also be done via PUT instead of GET; you just need to be positioned at the correct place in the file before you do the SET EOF. DO NOT try to SET EOF past the EOF - this bomb out with a DOS error.

CMD A$

Allows you to temporarily exit from your BASIC/S compiled code and execute a DOS command, and have control returned to your compiled program afterwards. Just set any simple string variable to the command you wish to execute, and then do a CMD A$. Be sure the command executed does not overwrite your code; compile your program starting at 7000H or higher to avoid this problem (or even at 56000!).

ON ERROR

A limited form of error trapping is possible with BASIC/S. In this form, you may trap for DOS errors only, not errors in BASIC or ROM processing. There is no ERL or RESUME in this form; all you can do is take some action based on the DOS error that occurred. First you establish your error trap routine with ON ERROR. Your ON ERROR statement MUST occur AFTER the line in your error trap routine you want to branch to; thus

```
50 ON ERROR GOTO 100
```

is no good since 100 comes later than 50. So your program would normally start out with a jump around the first line of your error trap, to your ON ERROR statement :

```
10 GOTO 40
20 A=ERR
```

(c)1982 by Breeze/QSD, Inc.

```
30 GOTO 2000:'main error routine at 2000
40 ON ERROR GOTO 20
```

BASIC/S must already KNOW where in memory your error routine will be when it encounters the ON ERROR statement; hence the requirement for the error trap to come before ON ERROR.
The very FIRST thing your error trap must do is set some integer variable to ERR, to grab onto the error code. If you wait to do this, ERR will change and not be relevant. Finally, the code returned in ERR is the same as the DOS error codes that are explained in your DOS manual, which are returned in register A whenever a DOS error occurs. For example, if ERR were 24, this would indicate 'File not found'.

ASSIGNMENT statement :

Following are the allowed forms of the assignment statement.

INTEGERS :

Integer arithmetic is limited to +,-,*,/ and only 2 operands allowed on the right hand side. No builtin functions for integers. Constants may be used, however.

Thus:

```
X=A*B
X=5-B
```

Note that unary minus is not allowed here (for variables) ie X=-Y+Z is no good, while X=Z-Y is OK. However, with constants you may use unary minus freely. Anything of the form X=AsB is OK, where A and B are integer variables or constants and s is one of +,-,*,/, as long as you don't have two minus signs adjacent.

STRINGS

```
A$=B$
A$="constant"
A$=B$+C$      (simple concatenation)
```

Also we have the builtin string functions ASC, LEN, CHR$, LEFT$, VAL, RIGHT$, MID$, STR$, and INSTR. Where numeric arguments are required in the string functions, simple integer variables or constants must be used - no expressions. The actual string arguments cannot be constants, but:

```
A$=LEFT$(X$,2)
```

(c)1982 by Breeze/QSD, Inc.

(for example) would be OK.

Also, expressions must be reduced to their simplest form -- e.g., concatenation within a function or function composition is not allowed. Break it down!

Note: The INSTR function differs from the regular DISK BASIC one in that no starting position may be specified -- syntax is just N=INSTR(A$,B$). However, unlike previous versions of BASIC/S, ALL of B$ is searched for, not just the first character.

MID$ note -- you can use MID$ on the left hand side of the = sign, and in that case, you can use either of the two forms MID$(A$,N)=B$ or MID$(A$,N,L)=B$ -- but they will give the same results, i.e. the length of B$ is used, L is ignored in the second form. If the source string (B$) is null, nothing is done.

Note III: The INKEY$ function is implemented, and must be used in the form: A$=INKEY$ (or B$, etc.).

HEX$

This is a hex conversion function, not supported by TRS-80 Disk Basic, but is supported under Microsoft BASIC-80 (and by their BASIC Compiler). BASIC/S II also supports it; what it does is to take an integer argument (variable or constant) and convert it to a hex string equivalent in value to the original integer. Thus

```
A$=HEX$(-1):PRINT A$
```

would print out "FFFF" (no quotes).

DISK I/O statements

Essentially, you have ten disk I/O buffers available for use (0-9), all of which may be used for sequential access, and two (1 and 2) of which may be used for random access. Here are the specifics :

OPEN

The OPEN statement is essentially that of disk BASIC, except that the filespec must be a string variable and not an expression in quotes. Syntax is

```
OPEN"m",b,F$<,r>
```

(c)1982 by Breeze/QSD, Inc.

where m = mode = I,O,R, or E
b = buffer = (0-9) (constant only)
(must be 1 or 2 for direct access)
F$ = filespec (variable only)
r = logical record length (optional -- may be either an integer constant or an integer variable).

BASIC/S makes few restrictions on your use of the disk I/O statements, so be careful. For example, if you wanted to open a sequential file with an LRECL of 16, you could. However, you would probably be well advised to stick to direct access files for this!

OPEN"E" is like OPEN"O" except you start out positioned at the end of the file.

Sequential I/O is done with the LINE INPUT# and PRINT# statements. Just specify a buffer number adjacent to the #, and you are ready to go. Only a simple string variable may be input or output, although PRINT#1,A$; will disable the carriage return.

Random disk I/O is accomplished via the following :

FIELD

You must field your buffer in order to communicate between your strings and the disk file being accessed. Syntax is :

FIELD 1,nn AS A$,mm AS B$, ...

-- the buffer can be 1 or 2, the strings can be any of A$ thru Z$ (no array references allowed here!), and the numbers 'nn', 'mm' etc. must be integer constants (1-255 -- 0 is not allowed). Also you can't really use a multiple FIELD stmts for the same file -- the second will override the first. Moreover, the statements to process a random access file must be statically nested -- i.e. do not GOSUB or GOTO a later line to FIELD a buffer and then return to do your LSETs and PUTs, etc. Just OPEN the file, FIELD the buffer, process it, and CLOSE it, without GOSUBS and GOTOS. (At least, don't branch anywhere outside the range of statements between the OPEN and CLOSE stmts).

LSET

To place your strings into the buffer prior to being

(c)1982 by Breeze/QSD, Inc.

PUT to the disk, use LSET. Thus

```
LSET A$=B$
```

where A$ is one of the strings mentioned in your FIELD statement. If LEN(B$) is less than that of the field variable A$, it will be filled out with spaces in the buffer. If greater, only the leftmost portion of B$ (for the fielding length of A$) will be in the buffer.

PUT

Syntax is PUT b,N where b is the buffer number (1 or 2) and N is any integer variable, containing the record number to be put. The record number variable is not optional.

GET

As in GET 1,R -- gets the Rth record from the disk file, and places its contents into the string variables mentioned in the FIELD statement.

LOF

The LOF function is implemented and syntax is

```
N=LOF(b)
```

where b is the buffer number (1 or 2 -- must be a constant). This returns the number of records in the currently open file with buffer b.

CVI and MKI$

For convenience in reading and writing integers from/to direct access files, these functions are implemented as in TRSDOS. In case you were mystified as to exactly what they did -- well, if the integer N has the 2 byte representation (L,H), then MKI$(N) is just CHR$(L)+CHR$(H). CVI just does the exact reverse. As with most BASIC/S functions, these may be used only with simple integer/string variables.

CLOSE

There is no global close in BASIC/S -- you must mention the buffer number. Thus,

(c)1982 by Breeze/QSD, Inc.

CLOSE 5

would close the file with buffer number 5. If you close a file that isn't open, you will bomb out with 'FILE NOT OPEN'.

EOF

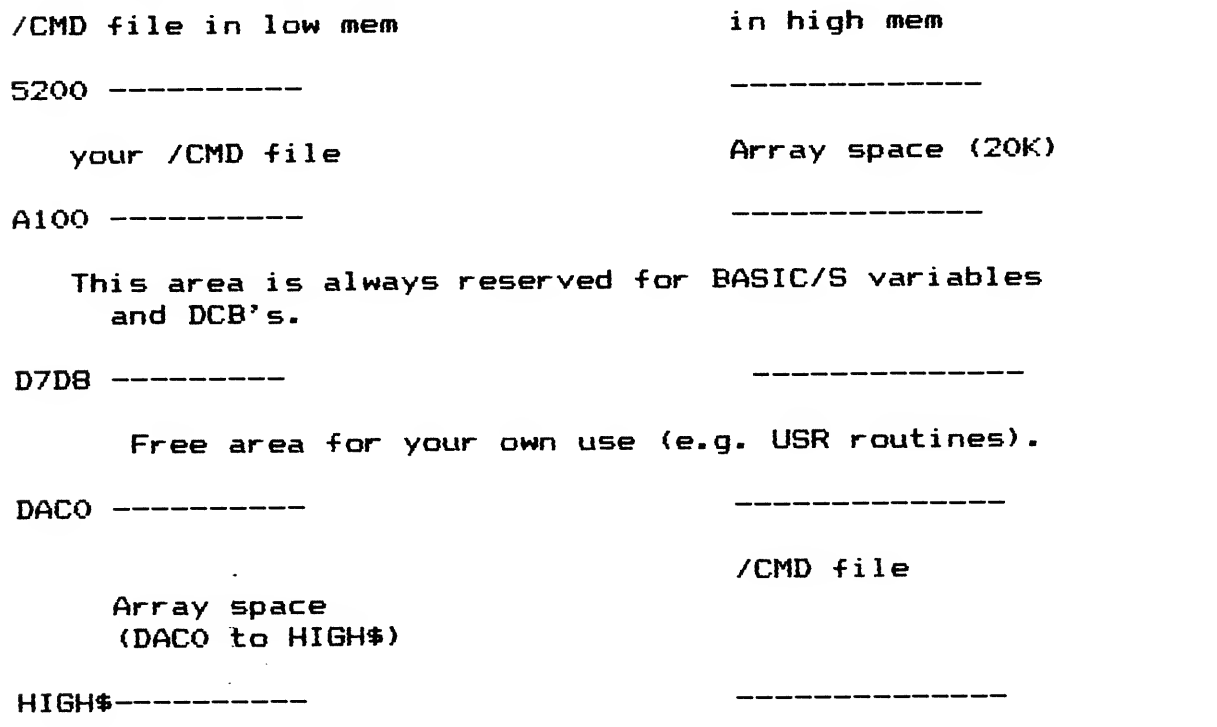
This isn't a function as such; it is to be used in a special form of the IF statement to check for EOF when inputting from a file. Simply say

IF EOF(b) THEN 200

(or whatever line number) to check for end of file on buffer b (0-9)

BASIC/S Memory Map

Following is a map of memory from 5200H up to HIGH$, showing how BASIC/S uses the memory in your TRS-80 (48K):



(c)1982 by Breeze/QSD, Inc.

--DISCLAIMER OF WARRANTIES & LIMITATIONS OF LIABILITIES --

We have taken great care in preparing this package. We make no expressed or implied warranty of any kind with regard to this manual or to BASICS/II. In NO event shall we be liable for incidental or consequential damage in connection with or arising out of the performance of this program.

BASICS/II (c)1982 by Bill Stockwell and Breeze/QSD, Inc.

All rights reserved. No part of this manual and NONE of the programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by information storage retrieval system, BBS, etc. Registered owners are entitled to make copies of the disk for their OWN use only!

Questions should be addressed to:

Bill Stockwell
4771 NW 24th #228N
Oklahoma City OK 73127
(405) 947-4156
Mnet 70070,320

Bill Stockwell may also be reached on the QSD Sig on MicroNet. Leave a message to 70001,610 for info or from the OK prompt, type R QSD<enter>.

Published by:
PowerSoft - a division of Breeze/QSD, Inc.
11500 Stemmons Expressway Suite 125
Dallas, Texas 75229

TRS-80 and TRSDOS are registered copyrights of the TANDY CORP.
LDOS is a registered trademark of Logical Systems, Inc.
Newdos and Newdos/80 are trademarks of Apparat
Dosplus is a trademark of Micro Systems Software